Effect Of Formal Specifications On Program Complexity And Reliability: An Experimental Study *

Amrit L. Goel†
Department of Electrical and Computer Engineering
Syracuse University
Syracuse, NY 13244
(315)443-4350
goel@suvm.acs.syr.edu

Swarupa N. Sahoo‡
School of Computer and Information Science

October 16, 1990

Abstract

In this paper we present the results of an experimental study undertaken to assess the improvement in program quality by using formal specifications. Specifications in the Z notation were developed for a simple but realistic anti-missile system. These specifications were then used to develop 2 versions in C by 2 programmers. Another set of 3 versions in Ada were independently developed from informal specifications in English. A comparison of the reliability and complexity of the resulting programs suggests the advantages of using formal specifications in terms of number of errors detected and fault avoidance.

*Funding during the course of this project was provided via US Army contract no. DAKF 11-89-C-0038 and NASA Grant no. NAG-1-806.

†Professor, ECE and CIS

‡Graduate Assistant, CIS

EXTENDED ABSTRACT

Specification languages are widely accepted as a stepping stone for design and development of a complex software system [1, 2, 3, 4, 5]. The advantages of a specification language are often not immediately clear in terms of program quality and reliability. Proving an executable program correct for complex systems is computationally an intractable task [6]. Also "an effective testing strategy which is reliable for all programs cannot be constructed" [7]. In such a setting, formal specification languages coupled with structured design methodologies [8] provide a streamlined approach for software design and development.

In this experimental investigation, we study the effect of the specification language Z [11] on program reliability and complexity. For our experiment we chose the NASA Launch Interceptor Problem(LIP) since it has been used extensively for several other studies in software reliability and fault tolerance. It is a simple but realistic representation of an anti-missile system. The original specifications were taken from Knight and Leveson [12]. The LIP is a constraint satisfaction problem, a solution to which is a decision procedure which takes a set of input points and launch characteristics to evaluate a set of initial launch conditions, called the preliminary unlocking matrix. The procedure then evaluates a logical combination(the combination is decided by an input matrix) of the initial conditions called the final unlocking vector the components of which collectively decide if the launch signal should be true or otherwise.

The experiment consisted of usual phases of software design and development with minor differences. In the specification phase a set of specifications of the requirements was developed in the Z notation and was validated by other specifiers. Several versions were developed based on informal and formal requirements specifications separately, by independent groups of programmers. For testing, a hybrid approach [13] was developed based on functional and structural information about the LIP. For generating test cases, the hypothetical launch conditions were divided into 7 relatively independent groups. The truth values of one of the groups was fixed a priori, and an input data set was constructed to satisfy the prefixed truth values of this group and the truth values of the rest were computed against the input set. Such manually designed test cases were used to test each program. After debugging, when the computations of launch conditions for all the versions match, the cyclomatic complexity measure [9] is applied to compute internal complexity of each individual module.

Also computed are the external complexity due to the interconnections between various modules based on "information flow" concepts [10], and finally the total system complexity as a weighted sum of internal and external complexities.

The versions based on informal requirements are found to be afflicted with usual problems caused by the inherent ambiguities in the informal requirements. However, a significant reduction was observed in the number of errors detected in the testing phase in case of the versions based on formal requirements. Further, complexity measures strongly suggest that versions based on formal specifications are less complex and more reliable than those based on informal requirements. The study also suggests that the formal specifications developed through several successive stages of operations refinement lend themselves to an automatic modular program development(special case of a divide and conquer technique) in an optimal way, and thus reduce the error-proneness of the program and make it more reliable.

Summary of Experimental Results

I. Productivity:

Table 1 - specification development time

Version number	Total Specification Development Time(hours)
Spec I	47

Table 2 - program development time

Version number	Total Program Development Time(hours)
Cver I	18
Cver II	38
Adaver I	76
Adaver II	73
Adaver III	89

II. Reliability*(in terms of number of errors detected)*

Table 3 - Number of errors detected during development

Version number	Total Number of Errors
Cver I	3
Cver II	8
Adaver I	8
Adaver II	7
Adaver III	4

Table 4 - Number of errors detected during testing

Version number	Total Number of Errors
Cver I	0
Cver II	7
Adaver I	13
Adaver II	11
Adaver III	8

References

[1] D. L. Parnas, *The Use of Precise Specification in the Development of Software*, Proceedings of IFIP Congress 77, Toronto, 1977.

[2] I. Hayes, editor. *Specification Case Studies.* Prentice-Hall International, London, 1987.

[3] C. Morgan, B. Sufrin, *Specification of the Unix Filing System*, IEEE Trans. Software Eng., March 1984.

[4] D. Bjorner, C. B. Jones. *Formal Specification and Software Development.* Prentice-Hall International, London, 1982.

[5] C. B. Jones, *Systematic Software Development using VDM*, Prentice-Hall International, 1986.

[6] D. L. Parnas, *When can Software be Trustworthy ?*, COMPASS-86 Conference, Washington, D. C., July 1986.

[7] W. E. Howden, *Reliability of the Path Analysis Testing Strategy*, IEEE Trans. Software Eng., Sep 1976.

[8] E. Yourdon, L. L. Constantine, *Structured Design.* Prentice-Hall Inc., 1979.

[9] T. J. McCabe, *A Complexity Measure*, IEEE Trans. Software Eng., Sep 1981.

[10] K. S. Lew, T. S. Dillon, K. E. Forward, *Software Complexity and its Impact on Software Reliability*, IEEE Trans. on Software Eng., Nov 1988.

[11] J. M. Spivy, *The Z Notation: A Reference Manual*, Prentice Hall International, 1989.

[12] J. Knight, N. Leveson, *An Experimental Evaluation Of The Assumption Of Independence In Multi-version Programming*, IEEE Trans. on Software Eng., Jan 1986.

[13] A. L. Goel, *An Experimental Investigation Into Software Reliability*, RADC-TR-88-213, Oct 1988.

VIEWGRAPH MATERIALS

FOR THE

A. GOEL PRESENTATION

EFFECT OF FORMAL SPECIFICATIONS ON PROGRAM COMPLEXITY AND RELIABILITY: AN EXPERIMENTAL STUDY

Amrit L. Goel[1]
Swarupa N. Sahoo[2]

Syracuse University
Syracuse, NY 13244

Presented at the Fifteenth Annual Software Engineering Workshop (SEL) held at NASA/G SFC, Greenbelt, MD, November 28-29, 1990.

[1] Professor, Electrical and Computer Engineering and School of Computer and Information Science, (315) 443-4350, goel@suvm.acs.syr.edu.
[2] Research Assistant

OUTLINE

- Objectives of Study
- Experimental Appraoch
- Results of Experiment
- Comparison with Versions from Informal Specifications
- Fault Aviodance by Using Z
- Concluding Remarks

OUTLINE

- Objectives of Study

- Experimental Appraoch

  - LIP Problem
  - Z-Specifications
  - Experiment Description

- Results of Experiment

  - Development Effort
  - Size and Complexity Metrics
  - Errors During Development
  - Errors During "Operational Testing"

- Comparison with Versions from Informal Specifications

- Fault Aviodance by Using Z

- Concluding Remarks

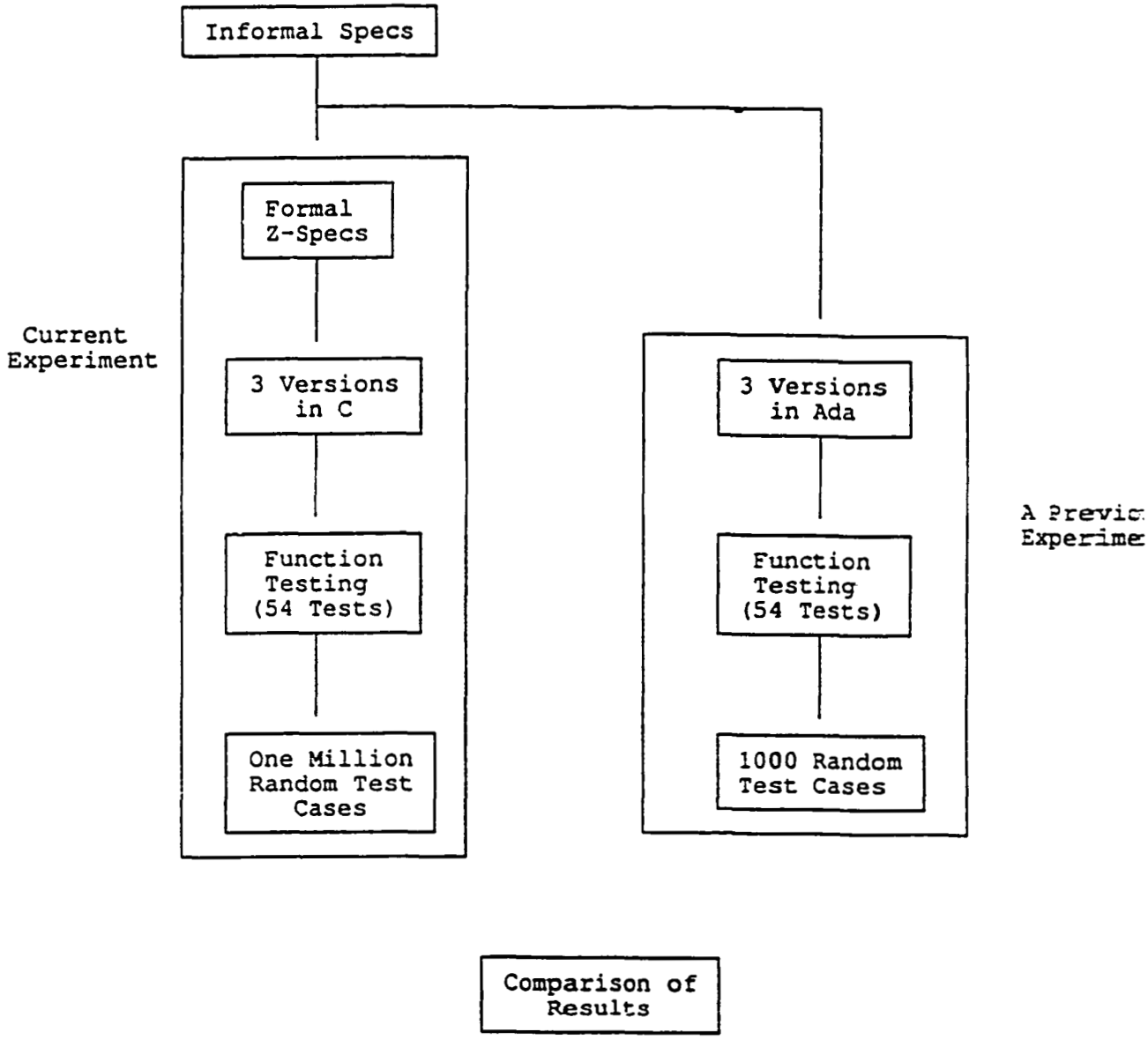
OBJECTIVES OF STUDY

- **Investigate the effect of using formal specifications on**

  - productivity
  - reliability
  - complexity

- **Compare results with versions developed from informal specifications**

EXPERIMENTAL APPROACH



EXPERIMENTAL APPRAOCH

- Used NASA - Launch Interceptor Problem (LIP)
- Developed Z-specifications from English specifications of LIP (Two independent Z specifications)
- Used Z-specs to develop 3 indipendent versions in C
- Each version tested for a set of 54 test cases from a previous experiment involving LIP
- Each version executed for one million random test cases to simulate operational testing

LIP

- Simple, but realistic anti-missile system.

- Studied elsewhere* in connection with fault-tolerant and Fortran/Ada comparison software research

- Program reads inputs which represent radar reflections, checks whether some prespecified conditions are met and determines if the reflections come from an object that is a threat and if yes, signals a launch decision

---

* Knight and Leveson, IEEE-TSE, January 1986.
Goel, etal, COMPSAC 87 and RADC-TR-88-213.

INPUT
Read Global values: NUMPOINTS, X, Y PARS.
Read Global Values: LCM, PUM-DIAGS
Evaluate LIC's
CVM
Compute PUM
PUM
Compute FUV
FUV
Compute LAUNCH
LAUNCH
CVM
PUM
FUV
LAUNCH
OUTPUT

EXAMPLE

Launch Intercepter Conditions

LIC 1: There exists at least one set of two consecutive data points that are a distance greater than LENGTH 1 apart

$$0 \leq = \text{LENGTH1}$$

LIC 11: There exists at least one set of three data points separated by exactly E and F consecutive intervening points, respectively, that are the vertices of a triangle with area greater than AREA1

$$1 \leq E$$
$$1 \leq F$$
$$E + F \leq \text{NUMPOINTS} - 3$$

Z-SPECIFICATION LANGUAGE

- Well known specification language developed by Programming Research Group at Oxford University

- Has been applied to develop specifications for several software systems but we are not aware of experimental results comparing it with informal approaches

SOME COMMENTS ON Z FOR LIP

Z-specifications were helpful in several aspects.

Some Examples:

- In resolving certain ambiguous issues

  - whether two identical (x, y) pairs can belong to a sequence of input data points

- In expressing invariant properties

  - the LCM matrix is symmetric, can be easily expressed mathematically

- In exploiting the repetitiveness of certain launch conditions which was helpful in functional groupings for design and testing.

  - a closer look at LIC 1, 8 and 13 indicates that they are related. We exploit the similarity by defining a "prototype" schema, and then using it to define each of these separately

Expressing Requirements in the Z Notation

Example:LIC1

Informal Specification

LIC 1: There exists at least one set of two consecutive data points that are a distance greater than LENGTH1 apart. (LENGTH1 $\geq$0)

Formal Specification

$LIC1[NUMPOINTS, LENGTH1]$
$POINTS : seq\ R \times R$ $cmv, cmv' : \mathcal{N} \rightarrow \mathcal{B}$
$cmv' = cmv \oplus$ $\{1 \mapsto (1 \leq \#\{(POINTS(i), POINTS(j)) \mid \forall i, j : 1..NUMPOINTS \bullet$ $j = i + 1 \wedge (LENGTH1 < edist(POINTS(i), POINTS(j)))\}$ $\wedge LENGTH1 \geq 0)\}$

where $edist(p, q)$ computes the distance between points p and q.

Expressing Requirements in the Z Notation

Example:LIC7

Informal Specification

LIC 7: There exists at least one set of N_PTS consecutive data points such that at least one of the points lies a distance greater than DIST from the line joining the first and last of these points. If the first and last points of these N_PTS points are identical, then the calculated distance to compare with DIST will be the distance from the coincident point to all other points of the N_PTS consecutive points. (DIST $\geq$0)

Formal Specification

$$
\begin{array}{|l}
\hline
LIC7[NUMPOINTS, N_PTS, DIST] \\
\hline
POINTS : seq\ R \times R \\
cmv, cmv' : \mathcal{N} \rightarrow \mathcal{B} \\
\hline
cmv' = cmv \oplus \\
\{7 \mapsto (1 \leq \#\{(POINTS(i), POINTS(j)) | \forall i, j : 1..NUMPOINTS \bullet \\
j = i + N\_PTS - 1 \wedge \exists k : i + 1..j - 1 \bullet \\
(pt\_cmp(POINTS(i), POINTS(j)) \\
\wedge (edist(POINTS(i), POINTS(k)) > DIST)) \\
\vee (\neg pt\_cmp(POINTS(i), POINTS(j)) \\
\wedge (pdist(POINTS(i), POINTS(j), POINTS(k)) > DIST))\} \\
\wedge DIST \geq 0)\} \\
\hline
\end{array}
$$

where $edist(p, q)$ computes the distance between points p and q , $pdist(p, q, r)$ computes the perpendicular distance from point r to the line through p and q and $pt\_cmp(p, q)$ returns a boolean value $true$ if p and q are identical, and otherwise $false$

A. Gael
Syracuse Univ.

Expressing Requirements in the Z Notation(contd.)

Example:LIC7

$$pdist : \mathcal{R}^2 \times \mathcal{R}^2 \times \mathcal{R}^2 \rightarrow \mathcal{R}^*$$

$$\forall p, q, r \in \mathcal{R}^2 \bullet$$

$$\neg(pt\_cmp(p,q)) \Rightarrow pdist(p,q) = \frac{2 \times \Delta area(p,q,r)}{edist(p,q)}$$

Note that the line must be well defined, i.e, at least the points on the line must not be identical. Obviously this is a partial function.

RESULTS OF EXPERIMENT

SOME PROGRAM METRICS

	C-Code From Z-Specs			Ada Code From Informal Specs		
Programmer	A	B	C	D	E	F
Source Lines	373	407	669	691	624	851
Comment Lines	82	80	59	59	126	251
System Complexity*	56	53	81	334	309	297

* See Lew et al, TSE, November 1988.

COMPLEXITY METRIC*

- System S has n modules, each with complexity M_i

- System complexity = $\sqrt{\Sigma M_i}$

- M_i depends on

  - Internal complexity
  - External complexity (measures module interrelationships)

- Internal complexity

  - McCabe's cyclomatic number

- External complexity

  - Amount of interaction with the environment
  - Depends on the information content of input and output parameters

* Lew et al, IEEE-TSE, November 1988, pp. 1645-1655.

PROGRAM DEVELOPMENT EFFORT (hours)

Versions	Develop Z-Specs	Design	Coding	Testing	Total
A	27	6	6	6	45
B	10*	10	10	8	38
C	33	8	6	4	51

*** B used specs. developed by A**

Learning Z: A - 20 hrs.
C - 21 hrs.

NUMBER OF ERRORS*

Programmer	Development and Unit Testing	Function Testing (54 TC)	"Operational" Testing (1 million TC)	Total
A	3	0	0	3
B	1	7	0	8
C	3	0	0	3

*Does not include compilation errors

COMPARISON OF DATA FROM C AND ADA VERSIONS

- We compared the effort and error data from a previous experiment that used Fortran and Ada languages.

- We do not think that our results are biased because language dependent aspects are not under study here. Also, the programmers in these studies were reasonably proficient in the respective languages so that the choice of the language should not affect our results

- However, to enhance our conclusions, we plan to develop C versions from informal specifications

COMPARISON OF EXPERIMENTAL RESULTS: EFFORT AND ERRORS

		Z			Informal		
Programmer		A	B	C	D	E	F
Effort		45	32	51	76	73	89
Errors	D&UT	3	1	3	5	4	4
	FT	0	7	0	8	7	4
	Total	3	8	3	13	11	8

D&UT - Development and Unit Testing

FT - Function Testing

FAULT AVOIDANCE BY USING Z

- We believe that certain types of faults can be avoided by using formal specifications

- Following are two explicit examples of faults avoided by using Z for LIP*

  - Caluclation of angle between π and 2π rather than between 0 and π

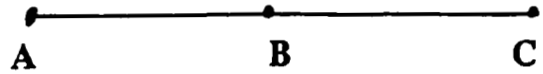
  - Calculation of distance from point to line when points are collinear and first point not between other two

---

* See Brilliant et al, TSE, February 1990, page 242.

FAULT-AVOIDANCE - EXAMPLE LIC 7

- Consider 3 collinear points (A, B, C) as shown



- Need to compute distance from B to line AC (LIC 7)

- Computation* from informal specs can lead to

  Dist(A, C, B) = min (dist(A, B), dist(B,C))

- However, formal specifications always compute zero, the correct result

---

* See Brilliant et al, TSE, February 1990, p. 242.

CONCLUDING REMARKS

- Use of Z specifications was clearly helpful in reducing errors (and hence increasing reliability)

- Based on a few metrics, it is also evident that the complexity of code developed from Z was also lower

- Total effort involved, including learning Z and development of Formal specifications, was comparable to the effort involved in developing versions from informal specifications

Yet ---

- This experiment does not provide conclusive evidence about the superiority of formal specification over informal ones

- Further investigation necessary to explore the feasibility and usefulness of Z for large problems

- Reusability of such formal specifications also needs to be investigated